# Documentation for Quantum.h and Quantum.c

Steven Andrews, © 2003
See the document "LibDoc" for general information about this and other libraries.

```
float psiHO(float x,int n,float k,float m);
complex Bracket(float *psib,float *psik,float *xr,int n);
float NormalKet(float *psi,float *xr,int n);
complex Hamiltonian(float *psib,float *v,float *psik,float *xr,int
     n,float mass);
complex Dipole(float *psib,float *psik,float *xr,int n,float q);
void EQMFschrod(phptr u,void *k,phptr dudt);
int TKFplotpsi(float t,phptr u,void *tkptr);
float *TimeEvolve(float *psi,float *ur,float m,float *xr,int n,float Dt);
int TKFdipcorr(float t,phptr u,void *tkptr);
void DipCorr(float *psib,float *psik,float *ur,float m,float q,float
     *xr,int n,float Dt,float *dip);
float EigenketAHO(float *xr,float *ur,float *psi,int n,float k,float m);
sptr DipTransform(float *dip,float dt,int nt,float wmin);
```

Requires: `<math.h>`, `<stdio.h>`, `"dynsys.h"`, `"Plot.h"`, `"Cn.h"`, `"math2.h"`, `"Rn.h"`, `"Constants.h"`, `"random.h"`, `"Spectra.h"`, `"Quantum.h"`

Example program: `VSEsim2.c`

Written 1/00. Routines have been moderately tested. Works with Metrowerks C.

These routines are for numerical computations of quantum mechanics, using wavefunction representations. Many functions use wavefunctions called `psi`, `psib`, or `psik`, where `b` and `k` stand for bra and ket wavefunctions; these are all complex vectors, using the `Cn.c` complex style. They also always require `xr` to be a real vector of corresponding $x$ values and `n` to be the number of real values in `xr` and the number of complex values in `psi`. Where units are required, all functions here use the small SI unit convention, used in `Constants.h`.

`psiHO` is the normalized harmonic oscillator wavefunction equation, where `x` is the position, `n` is the energy level (`n` is an integer ≥0), `k` is the force constant, and `m` is the mass.

`Bracket` does a Dirac bracket computation, <psib|psik>, returning the result. `Normalket` both normalizes a ket and returns the real factor by the wavefunction was multiplied. `Hamiltonian` computes the hamiltonian bracket, <psib|H|psik>, where H is $-h'^2/2\text{mass}\ \partial^2/\partial x^2 + v(x)$. `Dipole` computes the dipole bracket, for either state or transition dipole moments, <psib|□|psik>, where □=q$x$ and `q` is the particle charge.

`TimeEvolve`, `EQMFshrod`, and `TKFplotpsi` are all used for integrating Schrödinger's equations of motion for a particle in a one dimensional potential, using the `dynsys.c` integrators and other routines. In general, only `TimeEvolve` needs to be called from externally, although it may need some tinkering to get the desired output format. Call it with `ur` as the potential function (identical to `v` in `Hamiltonian`), `m` as the particle mass, and `Dt` as the amount of time to be evolved. These routines assume the `xr` vector is uniformly spaced, although it would be easy to generalize if desired. `EQMFshrod` takes in a wave function, as a field dependent phase point, and outputs its time derivative, using the collection of parameters in `k` for constants and for work space. `TKFplotpsi` plots the complex

wavefunction every 1 fs, showing the real part in black and the imaginary part in red. One trick these routines use to reduce unneccessary rotations of the wavefunction, which is generally irrelevent to the user, is to subtract the expectation energy from the input potential and then separately keep track of the overall phase factor using the first scalar element of the phase point structure. Here's the actual equation used by `EQMFschrod`:

$$\partial\psi'/\partial t = -ih'/2m\ \partial^2/\partial x^2\ \psi' + i/h'\ [V(x) - \langle\psi_0|H|\psi_0\rangle]\psi'$$

where $\psi'$ is the wavefunction, minus the overall phase factor, and $\psi_0$ is the initial wavefunction.

`DippCorr` is very similar to `TimeEvolve`, but it outputs a complex dipole correlation function, using the assumption that the bra is an eigenfunction of the potential. It uses `TKFdippcorr` to compute the correlation function during the integration. The result function, `dip`, needs to be pre-allocated before `DipCorr` is called, and it needs to be `Dt` complex elements in size (i.e. 2*`Dt`). Here's some math to show what it does, and how that was derived:

$$\begin{aligned} C(t) &= \langle\psi_b|\psi(t)\psi(0)|\psi_k\rangle \\ &= \langle\psi_b|U^\dagger(t)\psi U(t)\psi|\psi_k\rangle \\ &= q^2\ [\langle\psi_b|U^\dagger(t)]x[U(t)(x|\psi_k\rangle)] \\ &= q^2/N_{xk}\ \exp(-i\omega_b t)\ \langle\psi_b|x[U(t)(N_{xk}x|\psi_k\rangle)] \end{aligned}$$

Thus, the normalized wavefunction, defined as $x$ times the input ket, is time evolved using `EQMFschrod`, and the dipole correlation function is determined each femtosecond using this equation.

`DipTransform` inputs a complex dipole correlation function, the time step between each data point, the number of data points, and the minimum frequency, and returns a spectrum for it.

`EigenketAHO` uses the variational principle and a random search method to find the lowest energy eigenket for an anharmonic oscillator. It does this by adding the optimal amounts of many higher energy harmonic oscillator wavefunctions to the harmonic oscillator ground state wavefunction.